Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления

Кафедра технологий программирования

Попов Никита Алексеевич

Выпускная квалификационная работа бакалавра

Исследование задачи классификации веб-сайтов с учётом свойств зашумлённости и политематичности

Направление 010302

Прикладная математика и информатика

Заведующий кафедрой,
кандидат техн. наук,
доцент Блеканов И. С.

Научный руководитель,
доктор техн. наук,
доцент Печников А. А.

Рецензент,
кандидат техн. наук,
ведущий научный сотрудник Крижановский А. А.

Санкт-Петербург

2018

Содержание

Введение

В последние годы многократно возрос размер сети Интернет и, соответственно, количество информации в ней. В связи с этим востребованными являются задачи по автоматической обработке и классификации этой информации в общем и категоризации (структурировании схожих объектов по темам, формировании обобщающих множеств) веб-сайтов – в частности.

Одной из недостаточно изученных задач в этой сфере является классификация с учётом “зашумлённости” данных. Под “зашумлённостью” в данном случае будем полагать наличие веб-страниц сайта или их частей, которые слабо относятся к категории веб-сайта или не относятся к ней совсем (например, рекламные объявления (см. рис. 1) или разделы форума со свободной тематикой).

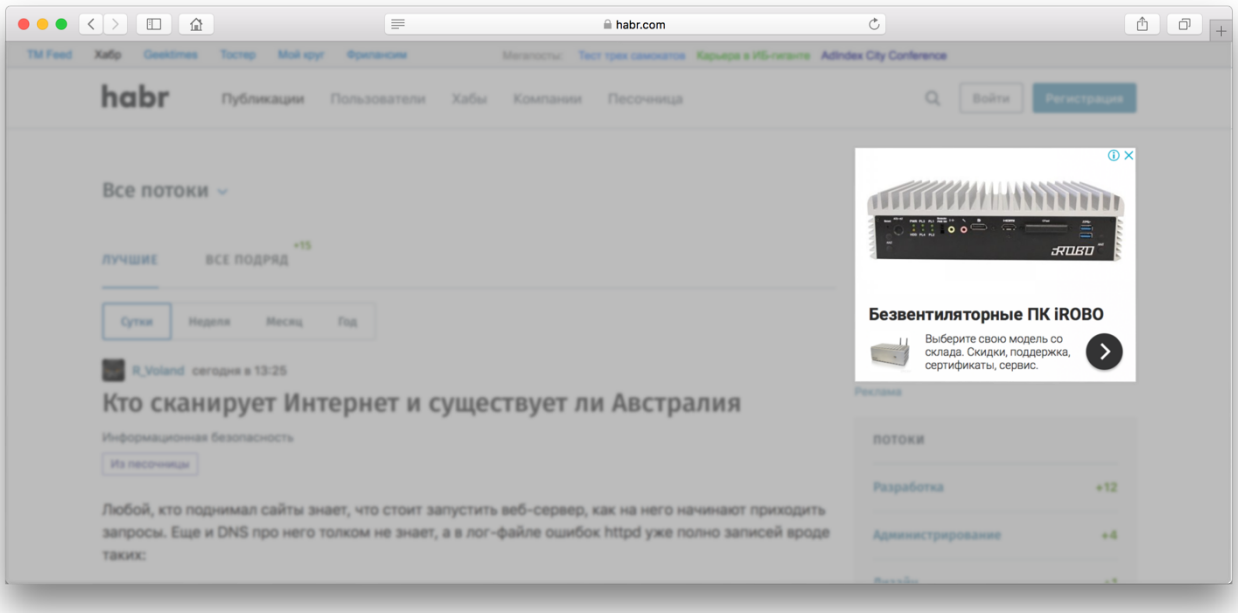

Рисунок 1. Пример рекламного объявления на веб-сайте

Параллельно с этим, актуальна проблема выделения академического веб-пространства из общего множества сайтов сети Интернет. Например, уже сейчас на оценку эффективности научных организаций влияет то, как она представлена в Вебе [1].

Для начала введём обозначения, которые будем использовать далее:

- веб-сайт – объединённая под одним адресом (доменным именем или IP-адресом) совокупность документов частного лица или организации;
- веб-страница – самостоятельная часть веб-сайта; документ, снабженный уникальным адресом (URL);
- hostname (имя хоста, доменное имя) – символическое имя, назначенное сетевому устройству, которое может быть использовано для организации доступа к этому устройству различными способами (например, для URL https://google.ru/ доменным именем будет google.ru);
- удалённость веб-страниц – характеристика, показывающая минимальное количество переходов со страницы А на страницу Б по ссылкам из тела веб-страниц;
- редирект – автоматическое перенаправление запроса с одного веб-адреса на другой;
- PageRank – числовая величина, характеризующая «важность», «авторитетность» веб-страницы;
- scraping (скрейпинг) – процесс извлечения данных из веб-страниц;
- слово в тексте – набор последовательно следующих символов определённого алфавита в тексте, отделенный от остальных символами пробела, табуляции, знаками препинания или иными специальными символами (например, в строке “3000 л.с. (эквивалентен паровозу УУ)” словами будут считаться “л”, “с”, ”эквивалентен”, ”паровозу”, ”УУ”);
- токенизация – разбиение электронного текста на отдельно значимые единицы (токены) для их последующей компьютерной обработки.

Работа организована следующим образом: во введении описаны предметная область, актуальность проблемы. Затем следует постановка задачи, а также основные этапы её решения. После этого приводится обзор литературы по данной проблеме и рассматриваются предлагаемые

варианты её решения. В первой главе производится исследование поставленной задачи и описываются различные методы и алгоритмы, предлагаемые для использования в работе, а также приводятся особенности их использования. Вторая глава посвящена алгоритму решения поставленной задачи. В третьей главе описана программная реализация классификатора и основные инструменты, применявшиеся при его реализации. В четвёртой главе описываются проведённые эксперименты и анализируются полученные результатов. В заключении подводятся итоги и предлагаются дальнейшие шаги по развитию данной работы.

Постановка задачи

В работе предлагается алгоритм для решения данной проблемы для веб-сайтов, относящихся к научно-образовательному Вебу (официальные сайты вузов Российской Федерации [2] и институтов Российской Академии Наук [3]). Одна из основных сложностей данной задачи состоит в том, что большая часть этих сайтов расположена в национальных доменных зонах (в нашем случае – *.ru, но это свойственно и многим сайтам университетов из других стран: https://mcgill.ca/, http://aalto.fi/), а не *.edu, специально предназначенной для подобных сайтов (например, https://stanford.edu/, https://mit.edu/). Таким образом, необходимо разработать модель классификатора, способного выделять подобные веб-сайты из множеств веб-сайтов произвольных категорий.

При этом речь идет не только об официальных сайтах вузов и институтов, которые широко известны и легко классифицируются по их URL. Более сложной и интересной задачей является задача классификации сайтов, имеющих невысокие рейтинги в поисковых системах, которые, тем не менее, важны для построения адекватных моделей научно-образовательного веб-пространства. К таким сайтам относятся сайты проектов, конференций, небольших научно-образовательных подразделений, персональные сайты учёных.

Также русскоязычное содержимое исследуемых веб-ресурсов накладывает значительное ограничение на выбор доступных инструментов, т.к. немногие из решений, доступных в свободном доступе, предоставляют средства для его обработки.

Цель работы – исследование задачи классификации веб-сайтов с учётом свойств зашумлённости и политематичности и разработка алгоритма ее решения.

Для достижения желаемого результата необходимо провести исследования и найти решения следующих подзадач:

- анализ существующих решений данной задачи; выявление их преимуществ и недостатков
- анализ данных веб-сайтов и выделение характеристик, необходимых для их успешной классификации; предварительная обработка данных веб-страниц;
- выбор метода машинного обучения для построения наиболее эффективного классификатора на выбранных характеристиках;
- определение оптимального числа страниц, необходимых для успешной классификации.

Обзор литературы

В работе [4] описан алгоритм классификации веб-сайтов, основанный на использовании ключевых слов, а также извлечении и обработке текстовых данных с веб-страниц и их последующей мультиклассификации. Также в работе показано, что использование в качестве характеристик информации о страницах, удалённых на 1 или 2 перехода (см. рисунок 2), слабо влияет на точность классификации. В результате сравнения различных методов классификации: k-Nearest Neighbors (kNN), Logistic Regression, Decision Tree, Bagging Random Forest и Support Vector Machine (SVM), последний показал наибольшую точность решения данной задачи.

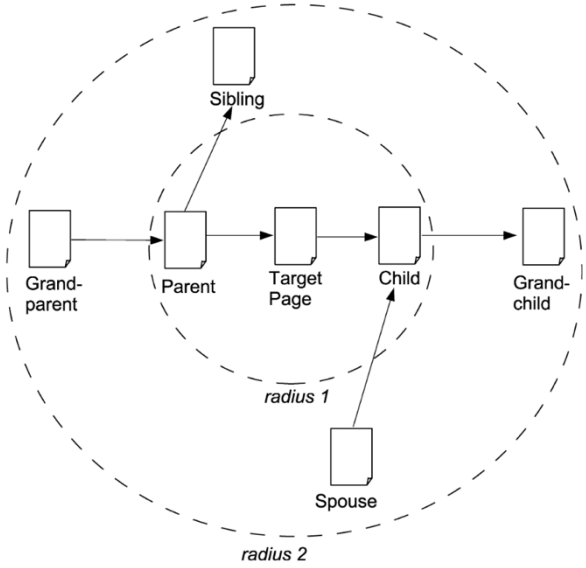

Рисунок 2. Виды веб-страниц, в зависимости от их расположения относительно исходной

В статье [5] описывается алгоритм классификатора, учитывающего свойства зашумлённости и политематичности данных веб-сайтов. Авторы отдельно отмечают, что основной целью является именно классификация веб-сайтов, а не отдельных веб-страниц, и приводят особенности данной задачи: необходимость учёта данных нескольких веб-страниц, а также возможность наличия веб-страниц сайта с тематикой, отличающейся от основной тематики сайта.

В работе [6] предлагается оригинальное решение данной задачи: для классификации веб-сайтов в качестве характеристик используются улучшенный алгоритм PageRank [7], применяемый на представлении веб-сайта в виде графа, и текстовое содержимое определённых тегов кода веб-страниц.

Статья [8] описывает алгоритм классификации веб-сайта с использованием заранее подготовленного словаря. Авторы подробно описывают процесс сбора и предобработки данных, при этом приводятся интересные решения, например, фильтрация списка url с помощью EasyList [9] перед обработкой.

Наиболее широко задача классификации веб-страниц рассматривается в работе [10]. В ней предложены различные характеристики веб-страниц, которые можно использовать для их описания, а также рассматриваются разнообразные алгоритмы классификации, производится их сравнение и приводятся области их применения.

В работе [11] рассматриваются 4 алгоритма обработки текстового содержимого веб-страниц, позволяющие составить их наиболее релевантное описание. Также в работе сравниваются два метода классификации: Naïve Bayesian Classifier и уже упомянутый SVM. Основным недостатком данной работы, как отмечают её авторы, является то, что в данной работе не учитываются свойства, которые можно извлечь из url веб-страниц. При этом рассмотренные алгоритмы показали повышение точности при применении к уже реализованным методам классификации более чем на 12%.

В работе [12] приводится описание поискового робота для сбора данных веб-страниц, основные идеи реализации которого, но несколько дополненные, использовались в данной работе. Также в работе описываются некоторые особенности сайтов научной направленности, рассматриваемых в настоящей работе.

В работе [13] описывается реализация морфологического анализатора pymorphy2, использующегося в данной работе. Отдельно рассматриваются особенности, учитываемые при работе со славянскими языками, а также подходы для работы со словами, отсутствующими в использующемся словаре.

Глава 1. Исследование задачи

В ходе анализа описанных выше работ были отмечены их следующие особенности (в том числе и недостатки):

- чаще всего задачу классификации веб-сайтов сводят к задаче классификации отдельных веб-страниц, без учёта зависимости между ними;
- в большинстве работ опускается описание процесса сбора данных с веб-ресурсов, несмотря на наличие некоторых отличительных особенностей этого процесса;
- под зашумлённостью данных обычно понимается наличие на веб-страницах различной рекламы.

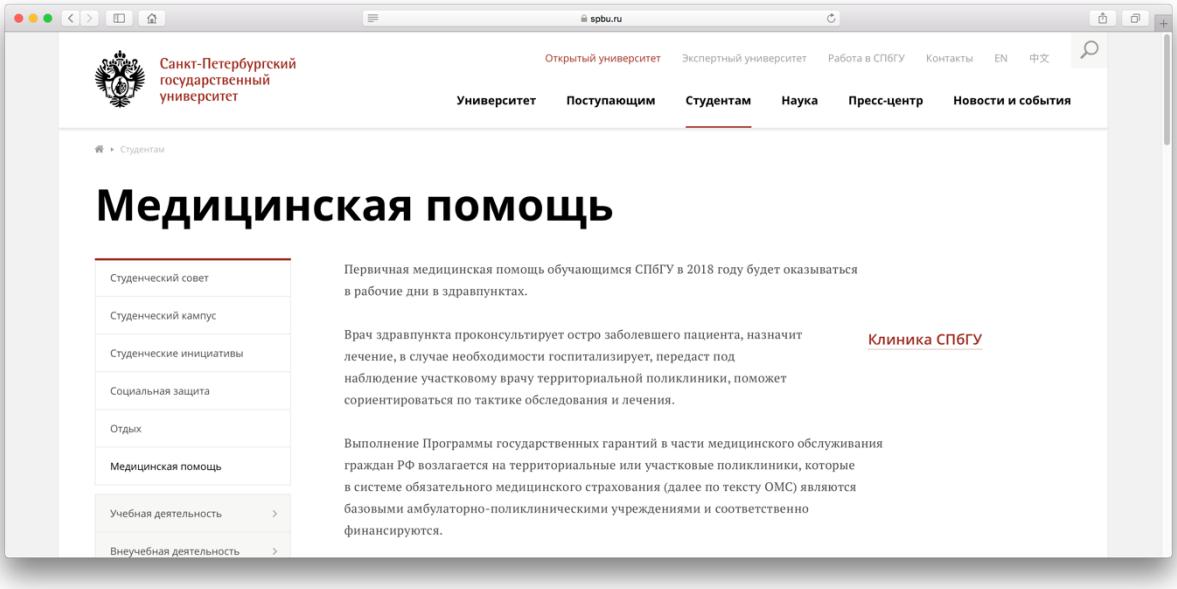

Рисунок 3. Веб-страница сайта СПбГУ, не относящаяся к описанию научной деятельности вуза

В данной работе приведённое выше понятие зашумлённости несколько расширено: дополнительно учитываются случаи, когда содержимое части веб-сайта (отдельных страниц или разделов) нельзя отнести к общей тематике сайта. Например, страницу веб-сайта СПбГУ, посвящённую медицинской помощи студентам вуза (https://spbu.ru/studentam/medicinskaya-pomoshch) сложно отнести к

категории «Научные учреждения» (см. рисунок 3).

В этом же примере проявляется и другая особенность: нередко при классификации веб-сайта к одной категории не учитывается факт того, что кроме основной тематики ресурса, значительная его часть может относиться к несколько иным категориям. В качестве наглядного примера можно привести веб-сайт селекционно-генетического центра http://indeikastav.ru/ (см. рисунок 4), в первую очередь содержащий информацию относительно коммерческой деятельности предприятия. При этом, значительная часть ресурса содержит информацию, посвящённую научной деятельности центра.

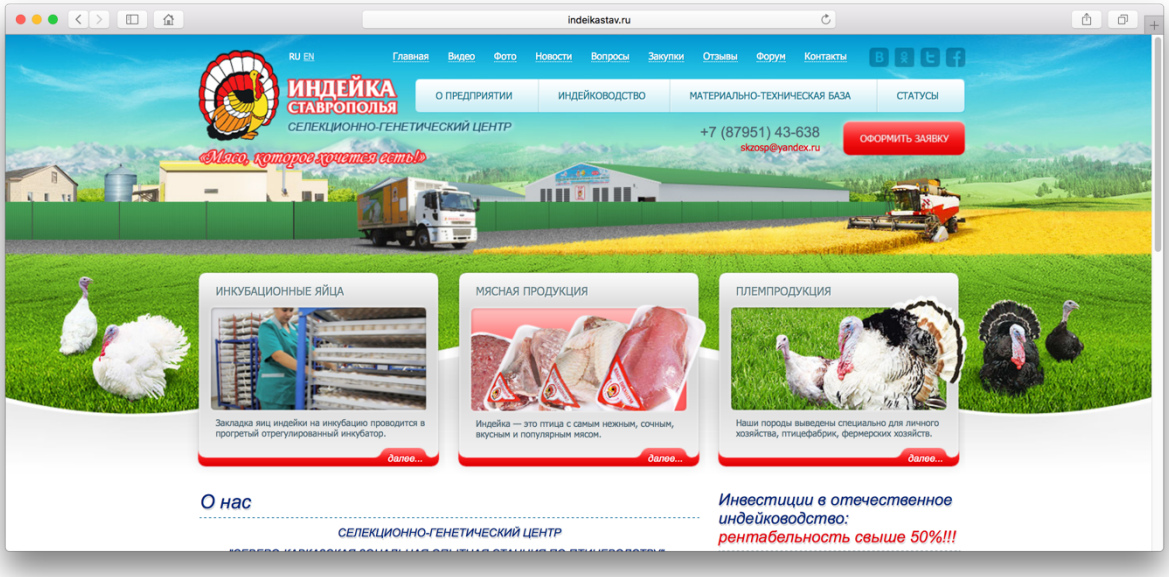

Рисунок 4. Главная страница http://indeikastav.ru/

Ещё один пример: веб-сайт http://pmpu.ru/ (см. рисунок 5), который не относится сайтам университетов, но авторами которого являются преподаватели факультета ПМ-ПУ СПбГУ. На данном ресурсе ими размещаются учебные материалы, связанные с читаемыми в университете курсами. В связи с этим было бы логично рассматривать этот сайт как часть научного пространства Веба.

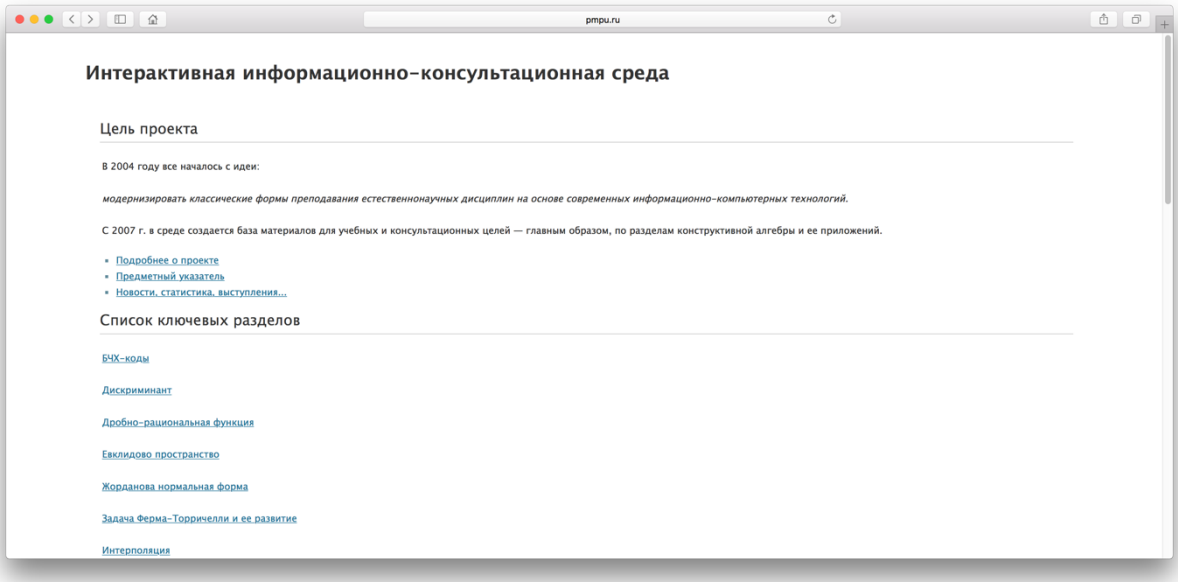

Рисунок 5. Главная страница http://pmpu.ru/

Учитывая эти факторы, было принято решение независимо классифицировать различные страницы веб-сайта, а затем принимать решение об отнесении веб-сайта к какой-либо одной категории. При этом, чтобы задача не сводилась к классификации веб-страниц без учёта связи между ними, дополнительно вводится правило, согласно которому выбираются страницы для обработки. Используемое правило обхода веб-сайта «вначале вширь» (также это метод называют “поиском в ширину”) [14] можно описать следующим образом: для корневого URL (страница нулевого уровня) получают список всех страниц данного сайта, доступных с этой страницы (будем называть их страницами первого уровня), затем повторяют это действие последовательно для всех страниц первого уровня (относя ко второму уровню лишь те страницы, ссылки на которые ещё не были получены ранее) и т.д. для каждого последующего уровня, пока не будет обработано необходимое количество страниц. Таким образом, для процесса классификации в первую очередь выбираются страницы «родственники» главной страницы веб-сайта, в общем случае наиболее релевантные категории веб-сайта [10].

1.1. Выбор характеристик

Для классификации веб-страниц могут использоваться различные характеристики, которые условно делятся на два типа: текстовые и визуальные [10]. Так как в поставленной задаче предполагается обработка данных сайтов научной направленности, решено использовать текст веб-страниц в качестве основного источника характеристик. Наиболее релевантными контексту страницы считаются содержимое заголовков страницы и её отдельных частей, а также различные мета-данные, предназначенные для обработки машинными алгоритмами. Для получения информации о структуре сайта и его связях с другими ресурсами используется информация, извлекаемая из ссылок, также содержащихся в тексте веб-страниц.

При этом, содержимое многих веб-сайтов зашумлено такими данными, как реклама, спам, фреймы (подключаемые области других веб-страниц), негативно влияющих на корректную классификацию ресурса. В различных расширениях для современных веб-браузеров для очистки содержимого веб-страниц от подобных данных используются специальные правила, на основании которых из кода страницы удаляются те или иные его части. Для русскоязычной части Веба наиболее популярными наборами таких правил (также их называют подписками фильтрации) являются RuAdList+EasyList и AdGuard Russian filter [15]. Их использование заметно снижает скорость сбора данных, т.к. последовательно анализируется содержимое кода всей страницы, но в результате качество этих данных значительно увеличивается.

Идея использовать PageRank в качестве одной из характеристик при классификации, описанная в [6], на данный момент (май 2018 г.) не представляется возможной, т.к. Google в 2016 году скрыл значения этой метрики из общего доступа [16].

В исследованиях по классификации веб-страниц на этапе обработки естественных языков обычно используется одна из следующих моделей представления текста [5, 11, 17]:

- мешок слов (bag of words);
- n-граммы (n-grams);
- latent Dirichlet allocation (LDA) и другие.

В данной работе для этой цели выбран мешок слов – модель текстов на натуральном языке, в которой каждый документ или текст выглядит как список, длина которого равна числу уникальных слов (токенов) в нашем словаре, а в каждом индексе в этом списке будет храниться, сколько раз данное слово встречается в предложении. Ячейка на пересечении строки и столбца содержит количество вхождений слова в соответствующий документ. Пример bag of words для следующего набора документов представлен в таблице 1:

1. Джон любит смотреть фильмы.
2. Ещё Джон любит смотреть футбол.

Таблица 1. Пример bag of words для набора документов

Уникальное слово		Джон	любит	смотреть	фильмы	ещё	футбол
Число вхождений слова в предложение №	1	1	1	1	2	0	0
	2	1	1	1	0	1	1

Основным недостатком такой модели является игнорирование порядка слов в предложении (поэтому она и называется «мешком»), из-за чего не представляется возможным извлечь контекст из предложений и абзацев. Также, разные формы одного слова в данной модели будут учитываться как различные токены, что отрицательно влияет на качество модели.

Обычно для минимизации влияния этого недостатка проводят предобработку текста одним из следующих методов: стеммингом или

лемматизацией с предварительной очисткой документа от так называемых стоп-слов (списка слов, часто встречающихся в тексте вне зависимости от его принадлежности к какой-либо категории). Стемминг – процесс нахождения основы слова для заданного исходного слова. При этом, основа слова необязательно совпадает с морфологическим корнем слова. Лемматизация – процесс приведения словоформы к лемме — ее нормальной (словарной) форме. В данной работе используется алгоритм нормализации, предложенный в работе [13], т.к. в результате его применения в один токен преобразуется большее, по сравнению с описанными методами, число различных форм одного слова благодаря использованию словаря и альтернативного алгоритма «предсказания» для несловарных слов.

Альтернативой описанной модели выступает рассмотрение в качестве характеристик веб-страниц n-грамм – последовательностей из n слов. Например, биграммами для первого предложения из примера выше будут являться “Джон любит”, ”любит смотреть”, ”смотреть фильмы”. Достоинством этого подхода является тот факт, что использование последовательностей слов позволяет получить представление о контексте словосочетаний. Стоит отметить, что bag of words содержит в себе n-граммы (n = 1, униграммы), но они лишены указанного достоинства в силу очевидных свойств.

При этом, использование n-грамм влечёт за собой увеличение размера словаря и, как следствие, объёма используемой памяти, а в некоторых случаях вызывает ухудшение качества модели [17], поэтому в данной работе решено использовать bag of words. Однако в дальнейшем следует провести сравнительные эксперименты для двух моделей с целью выбора лучшей.

1.2. Модель классификатора

В задаче классификации веб-страниц при использовании их текста в качестве набора характеристик одним из наиболее точных методов машинного обучения классификатора считается Support Vector Machine (SVM) [4, 11]. Он основывается на предположении о том, что наилучшим способом разбиения на классы множества точек в m-мерном пространстве является m-1 плоскость. Для этого производится перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве (см. рис. 6).

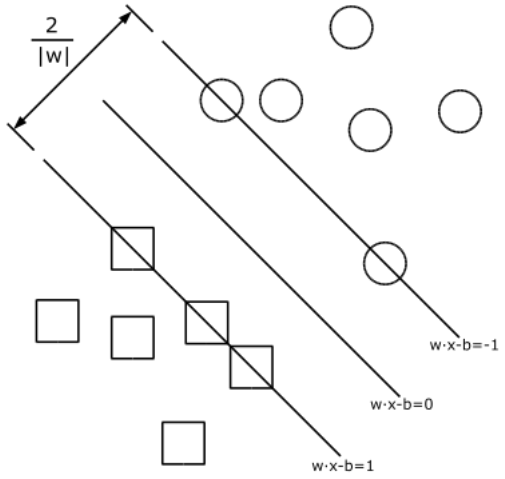

Рисунок 6. Пример разбиения множества на классы с помощью SVM

Для оценки качества классификации применяются метрики, использующие значения числа документов, отнесённых к той или иной категории (см. таблицу 2).

Таблица 2. Основные категории документов для оценки классификации

		Объект отнесён к классу классификатором?	
		да	нет
Объект относится к классу?	да	True Positive (TP)	True Negative (TN)
	нет	False Positive (FP)	False Negative (FN)

Базовыми метриками для оценки качества проведённой классификации считаются recall (полнота) и precision (точность) [18].

Полнота является оценкой вероятности того, что классификатор отнесёт веб-сайт к классу при условии, что сайт к классу действительно принадлежит, точность – что сайт действительно принадлежит классу в случае, когда классификатор отнесёт к нему этот сайт.

$$recall = \frac{TP}{TP + FP}$$

$$precision = \frac{TP}{TP + TN}$$

Также их обычно дополняют accuracy (аккуратность) и F-мера, соответственно отображающие долю правильно классифицированных веб-сайтов и среднее гармоническое precision и recall:

$$accuracy = \frac{TP + FN}{TP + TN + FP + FN}$$

$$F = \frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$

Глава 2. Алгоритм

Для решения поставленной задачи предлагается следующий алгоритм. Входные данные скрипта представляют собой число страниц веб-сайта для анализа и данные о рассматриваемом наборе веб-сайтов: тип (*train* для сайтов из обучающего множества, *test* – из проверочного), заранее присвоенную категорию (например, *university*, *institute*, *other*) и ссылку (url) на главную страницу (корень). В результате работы программы требуется получить список веб-сайтов, относящихся к типу *test*, для каждого из которых указывается значение категории, к которой он был отнесён построенным классификатором, а также подсчитать значения метрик, наглядно демонстрирующих качество реализованного алгоритма.

2.1. Извлечение данных о веб-сайтах

Для заданного количества страниц каждого сайта по правилу “вначале вширь” веб-краулером загружаются следующие данные: фактический url (с учетом переадресаций и протокола), содержимое заголовка страницы (данные тега <title>), мета-данные (содержимое атрибутов description, keywords и content тегов <meta>), а также весь текст, доступный посетителю веб-страницы (для этого с помощью парсера извлекается содержимое “тела” страницы, тега <body>), очищаемое при этом от скриптов и стилей (данные тегов <script>, <style>), а также различных “зашумляющих” данных (с помощью правил, используемых в контент-фильтре AdBlock Plus [19] для очистки русскоязычных страниц от рекламы, встраиваемых элементов социальных сетей и т.д.). Предусмотрена обработка различных ответов сервера (обрабатываются только ответы на запросы с кодом менее 400, соответствующим ответу сервера без ошибки [20]) и извлечение и анализ ссылок в тексте страницы (удаление нерабочих ссылок и ссылок на не-html содержимое; замена относительных ссылок абсолютными; отбрасывание

ссылок, ведущих не на ресурсы анализируемого веб-сайта и т.д.).

2.2. Предобработка текстовых данных

Полученные текстовые данные обрабатываются с помощью морфологического анализатора: производится очистка текста от лишних символов (знаки препинания, цифры, некириллические символы, слова длиной менее трёх букв и т.д.); удаление слов, слабо влияющих на классификацию текста из-за их роли связующих элементов в тексте (вместо использования списка стоп-слов в данной работе решено удалять все слова, относящиеся к следующим частям речи: числительные, предлоги, союзы, частицы, междометия, компаративы и предикативные наречия) и нормализация. Наконец, производится токенизация текста, заголовка и мета-данных: представление каждой веб-страницы в модель bag of words, готовую для обработки методами машинного обучения и последующего анализа.

2.3. Обучение классификатора

Для задачи классификации веб-сайтов выбран метод ключевых слов, подразумевающий оценки важности слов в наборах характеристик с помощью хеширующего векторизатора, довольно эффективного на практике и экономичного для ресурсов. На построенной модели обучается LinearSVC классификатор.

Процесс классификации веб-сайта представляет собой обработку каждой страницы обученным классификатором, после чего для каждого веб-сайта выбирается категория, к которой отнесено наибольшее количество страниц.

Глава 3. Разработка программы

В данной главе описывается структура реализованного программного решения, основные инструменты разработки и сторонние библиотеки, применявшиеся в ходе исследования. Отдельно приводятся особенности реализации методов и алгоритмов программы, а также механизмов хранения обрабатываемых и анализируемых данных.

3.1 Инструменты разработки

Основным языком разработки был выбран Python 3.6 из-за ряда преимуществ для решения задачи классификации, выделяющих его среди аналогов. Основными из них являются наличие различных библиотек с богатым функционалом для обработки текста и веб-сайтов, а также анализа данных. Основной средой разработки была выбрана PyCharm Professional.

Для обработки данных веб-сайтов используются встроенная библиотека urllib, предлагающая широкие возможности по обработке url адресов, а также сторонняя библиотека requests, предоставляющая богатый функционал для совершения запросов на веб-серверы и обработки ответов на них. Веб-краулер для обработки html-кода веб-страниц реализован с помощью функционала библиотек MechanicalSoup [21] и BeautifulSoup4 [22], предоставляющих необходимые средства для его реализации, при этом, оставляя разработчику широкие возможности по модификации полученного решения. С помощью встроенного пакета cssselect к коду применяются правила AdBlock.

Для хранения и взаимодействия с загруженной информацией используется сторонняя библиотека pandas [23], отличающаяся эффективными методами работы с большими объёмами данных.

Для лемматизации полученного текста применяется морфологический анализатор pymorphy2, использующий словарь

OpenCorpora и предлагающий широкие возможности по обработке текста. Помимо обширного словаря, библиотека предоставляет возможность анализа слов, не входящих в него, благодаря предусмотренному предсказателю. Один из основных недостатков анализатора – отсутствие возможности учитывать контекст при выборе нормальной формы слова.

Алгоритмы машинного обучения и средства анализа результатов классификации реализуются средствами библиотеки scikit-learn [24]. Также эта библиотека предоставляет функционал для объединения нескольких наборов характеристик с помощью классов Pipeline и FeatureUnion. Графические результаты выводятся с помощью функционала библиотеки plotly [25], предоставляющего большие возможности для построения различных графиков.

Некоторые из перечисленных библиотек написаны частично на языках программирования C и C++, обеспечивающих им быстродействие на ресурсоёмких задачах. Для асинхронных запросов используются средства языка для параллельных вычислений, что также заметно повышает скорость работы.

Реализованный алгоритм представлен в виде python скрипта, благодаря чему решение является кроссплатформенным и может запускаться на системах с ОС Windows, macOS и других системах семейства *nix.

3.2 Структура программы

Программное решение состоит из основного скрипта main.py, в котором описан весь алгоритм, и вспомогательных пакетов, содержащих описание используемых методов для различных задач.

В пакете init инициализируются основные переменные (путь к текстовому файлу с начальными данными, url адреса правил AdBlock, пути к файлам для вывода результатов и т.д.) и содержатся методы,

предназначенные для:

- загрузки исходных данных скрипта (`from_file(file), adblock()`)
- использования средств параллельных вычислений (`parallel(func, parameters, mode='map', threads=cpu_count() - 1)`)
- вывода результатов и уведомлений (`get_output(output, results), confusion_matrix(output, confusion_matrix), notify(title, text, subtitle='', sound='Glass')`)

В качестве файла с начальными данными принимается текстовый файл со следующей структурой (см. рисунок 7): первая строка содержит число страниц каждого веб-сайта, подлежащих анализу, затем построчно информация о каждом веб-сайте в формате: *{тип} {категория} {url корня сайта}*, где *{тип}* – название множества, к которому относится веб-сайт при классификации (train для обучающего, test – проверочного), *{категория}* – название категории сайта (описание категорий, используемых в данной работе приведено ниже), служащее для валидации результатов, *{url корня сайта}* – URL, служащий для доступа к странице, с которой начнётся анализ веб-сайта.

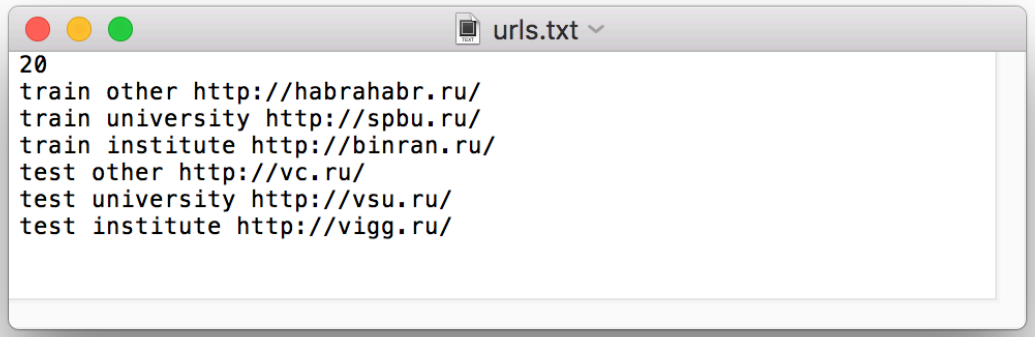

Рисунок 7. Пример текстового файла с входными данными

Пакет scraper предоставляет методы, необходимые для:

- получения актуальной ссылки для заданного url с учётом редиректов

(перенаправлений запроса) и протокола (`get_actual_url(url)`)

- извлечения следующих данных о веб-странице: актуальной ссылки, заголовка, мета-данных, текста, а также списка всех ссылок с данной страницы (`parse_url(url)`)

В пакете adblock описывается класс AdRemover с двумя методами:

- функция инициализации класса с заданными ранее правилами очистки (`__init__(self, *rules_files)`)
- функция очистки дерева тегов веб-страницы от удовлетворяющих правилам узлов (`remove_ads(self, tree)`)

Пакет manager содержит функционал для управления очередью адресов веб-страниц для анализа и добавления в них новых (`manage(queue, roots, dataframe)`), а также следующих вспомогательных методов:

- присвоение ссылке статуса обработанной, добавление её извлечённых данных в датасет, обновление в очереди числа страниц данного веб-сайта, необходимых для загрузки (`scrape(url, queue, roots)`)
- исправление ссылки на актуальную с получением информации о корне веб-сайта и его категории (`fix(child, roots)`)
- приведение ссылки в соответствие стандартному виду url с учётом ссылки предыдущего уровня (`fix_url(url, root)`)
- проверка url на доступность (`validate_url(url)`)
- извлечение данных о корне веб-сайта из url (`get_root(url)`)

Методы для обработки текстовых данных веб-страниц описаны в пакете text. Среди них:

- предобработка и токенизация (`parse_text(dataframe, input, output, engine='pymorphy')`)
- очистка текста согласно описанным выше правилам (`clear_text(text)`)
- удаление из текста частей речи, слабо влияющих на классификацию (`my_tokenizer(s, morph)`)

Функционал, необходимый для реализации классификатора, предоставляется пакетом classification. Для извлечения необходимых столбцов из DataFrame при использовании Pipeline реализован класс ItemSelector с двумя методами, требуемыми данной структурой: `fit(self, x, y=None)` и `transform(self, data)`. Также в пакете реализованы следующие методы:

- подготовка датасета к процессу обучения классификатора: разбиение на обучающее и тестирующее множества, вывод полученных результатов (`classify(dataframe)`)
- обучение классификатора с сохранением полученной модели в model.pkl и его проверка на тестирующем множестве (`predict(train, test, model)`)
- присваивание тестируемым веб-сайтам категории на основе полученных значений для каждой веб-страницы (`assert_class_to_root(dataframes)`)

3.3 Описание практической реализации алгоритма

Далее приводится описание реализованного алгоритма (см. рисунок 8) и некоторые его особенности. Актуальный исходный код доступен по адресу https://github.com/nikitalpopov/bachelor и в приложении 1.

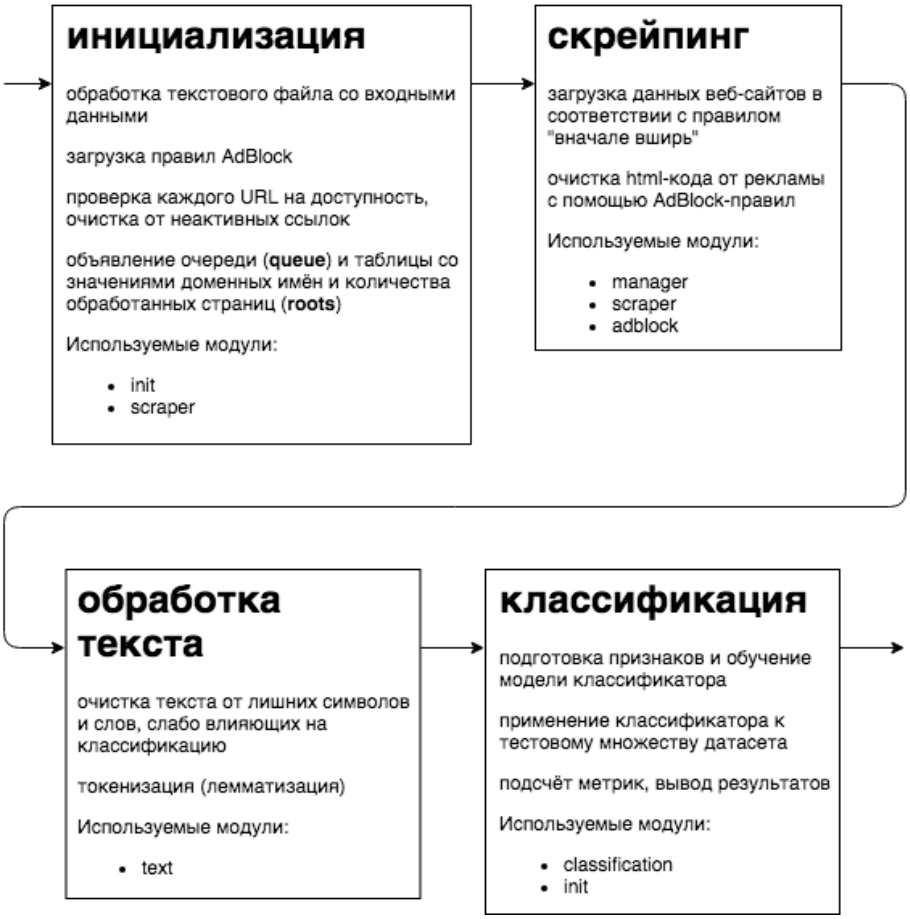

Рисунок 8. Наглядное представление алгоритма

Вначале алгоритм с помощью метода init.from_file() считывает число страниц каждого веб-сайта для анализа и описание каждого сайта датасета из текстового файла, путь к которому задаётся переменной init.URLS, в переменные children_count (целое число) и initial_data (pandas DataFrame со столбцами purpose (тип), category (категория) и url (ссылка на главную страницу веб-сайта), а также объявляется переменная init.CATEGORIES, содержащая все уникальные значения столбца categories из initial_data.

Затем методом scraper.get_actual_url() производится проверка доступности каждого URL и очистка исходного набора от неактивных

ссылок, а также замена исходных ссылок значениями, полученными в ходе редиректов. Полученные значения будут считаться корнями веб-сайтов. Также на этом этапе объявляются два pandas DataFrame: roots (таблица со столбцами purpose, category, root – доменное имя веб-сайта, children – количество страниц веб-сайта, которое необходимо обработать, инициализируется значением children_count) и queue (purpose, category, url, status – статус веб-страницы: ‘+‘, если обработана скриптом, ‘–‘ – иначе).

На следующем шаге методом manager.manage() производится обращение к заданным корням веб-сайтов и начинают обрабатываться данные для обучающего множества. Для каждого URL из очереди, относящегося к соответствующему веб-сайту, производится загрузка html-кода страницы с помощью метода manager.scrape(). Если содержимое ответа на запрос непустое, то значение roots[‘root’] для соответствующего корня уменьшается на единицу. Затем из кода каждой обработанной на данном шаге страницы извлекаются все ссылки и «исправляются» (удаляются неактивные, либо возвращающие на запрос ответ со значением кода менее 400; относительные ссылки преобразуются в абсолютные) с помощью manager.fix(). После этого удаляются повторяющиеся значения, каждой полученной ссылке присваивается статус ‘–‘ и обновляется очередь. Наконец, если для каких-то веб-сайтов число обработанных страниц будет меньше заданного, то описанный в данном абзаце алгоритм повторяется (т.е. происходит спуск на один уровень для правила «вначале вширь») до тех пор, пока условие не будет выполнено для каждого веб-сайта, либо не опустеет очередь.

Полученные текстовые данные (содержимое столбцов text, title и meta из DataFrame) очищаются от лишних символов средствами text.clear_text() и обрабатываются методом text.parse_text(), с помощью которого они нормализуются и сохраняются в файл. При этом удаляются слова длиной менее 3 символов, а также части речи, указанные выше, как слабо влияющие

на классификацию документа.

Наконец, методом classification.classify() производится построение структуры Pipeline: с помощью хеширующего векторизатора и преобразования полученных векторов признаков (текст, заголовок, мета-данные) в соответствие с таблицей токенов получаются данные для обучения модели. Затем производится обучение классификатора, модель которого сохраняется в файл model.pkl для возможности последующего её использования без повторения предыдущих шагов алгоритма процесса, после чего производится классификация каждой веб-страницы из анализируемого множества. Полученные значения используются для классификации веб-сайтов: производится подсчёт количества страниц, отнесённых классификатором к каждой из категорий и выбирается та, к которой относится наибольшее число.

Полученные результаты сохраняются в classification.xlsx файл и выводятся в окно консоли. Дополнительно производится подсчёт различных метрик для анализа качества классификации и строится confusion matrix для наглядного представления соотношения результатов классификации к реальным значениям.

Глава 4. Эксперименты и выводы

Для проведения экспериментов был собран датасет (см. таблицу 3), включающий три категории веб-сайтов. Сайты научной направленности условно разделены на две категории и включают официальные сайты вузов РФ официальные сайты институтов РАН (соответственно, категории *university* и *institute*), классифицированные в ходе конференций BOINC:FAST 2015 и 2017 годов [26]. Третью категорию *other* составляют веб-сайты из дорожки классификации веб-сайтов конференции РОМИП 2009 года [27].

Таблица 3. Описание датасета

Категория	**university**	**institute**	**other**
Описание	официальные сайты вузов РФ	официальные сайты институтов РАН	веб-сайты дорожки РОМИП 2009 года
Всего веб-сайтов	292	401	901
Веб-сайтов не обрабатывается (ссылка неактуальна)	21	21	199
Тестовых веб-сайтов	89	89	76

При составлении датасета использовались некоторые ресурсы, которые, как оказалось, на данный момент уже недоступны. Особенно это заметно для категории *other*, что во многом связано с «возрастом» источника информации о доменах: с 2009 года срок действия некоторых из них уже истёк, и теперь не представляется возможным получить данные, находившиеся там ранее. Также, реализованный веб-краулер счёл некоторые URL адреса как неактивные, что, скорее всего, связано с защитой веб-сайтов от автоматической обработки подобными алгоритмами и требует дополнительного исследования в будущем. Примером такого URL является http://tuvsu.ru/.

Кроме того, в датасете есть сайты, срок действия доменного имени которых истёк, и ресурсы сменили URL, например, http://chgpi.ru/. При

этом, на запрос по данному адресу возвращается ответ с кодом 200 и сообщение, предлагающее купить данный домен (см. рисунок 9). Стоит отметить, что, как и ожидается, подобные ресурсы относятся классификатором не к научным веб-сайтам (для конкретного ресурса предполагалось, что на нём расположен веб-сайт Чеченского государственного педагогического института), а к категории *other*.

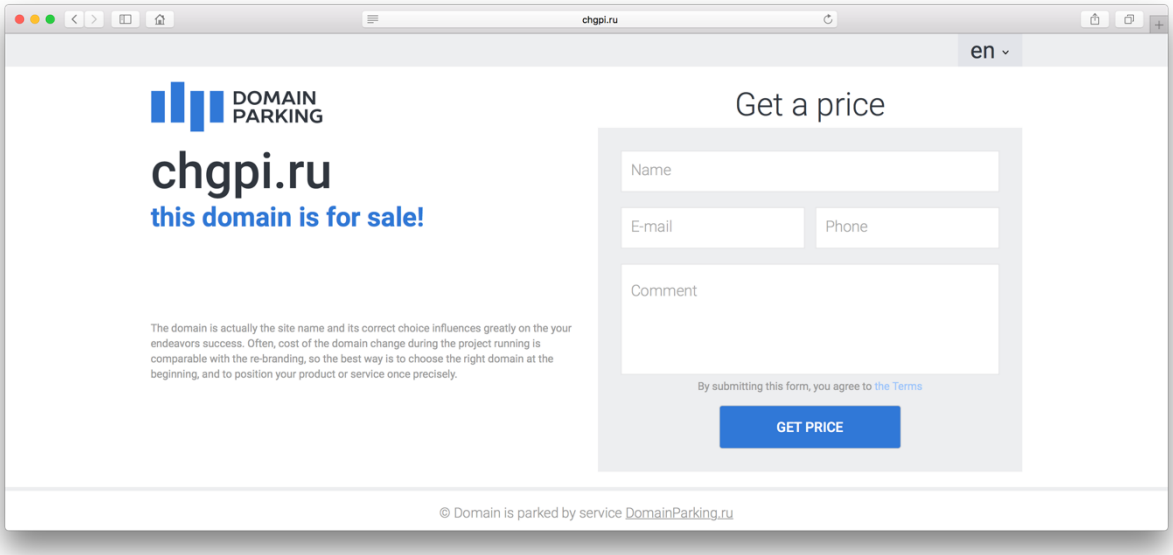

Рисунок 9. Страница, доступная по адресу http://chgpi.ru/

В ходе исследования для каждого веб-сайта было установлено ограничение числа загружаемых страниц для обработки в 20 штук. Это число было выбрано в ходе тестов с использованием различных значений (5, 10, 20, 50), т.к. при этом значении достигается приемлемая точность классификации и умеренно расходуется память устройства. Увеличение этого значения требует доработки алгоритма в части работы с памятью, либо использования устройства с большими доступными ресурсами. В таблице 4 приведены примеры классификации различных веб-сайтов датасета с указанием числа страниц, отнесённых к каждой категории:

Таблица 4. Примеры классификации различных сайтов из тестового множества

Веб-сайт (доменное имя)	Заданная категория	Присвоенная категория	Количество страниц, отнесённых к категории		
			university	institute	other
bsu.ru	university	university	20	0	0
apmath.spbu.ru	university	university	4	1	0
spbu.ru	university	university	9	0	0
sstu.ru	university	university	14	0	3
rniish.ru	institute	institute	0	16	4
indeikastav.ru	institute	other	0	4	6
wniipo.ru	institute	other	0	3	7
ihst.ru	institute	institute	0	8	2

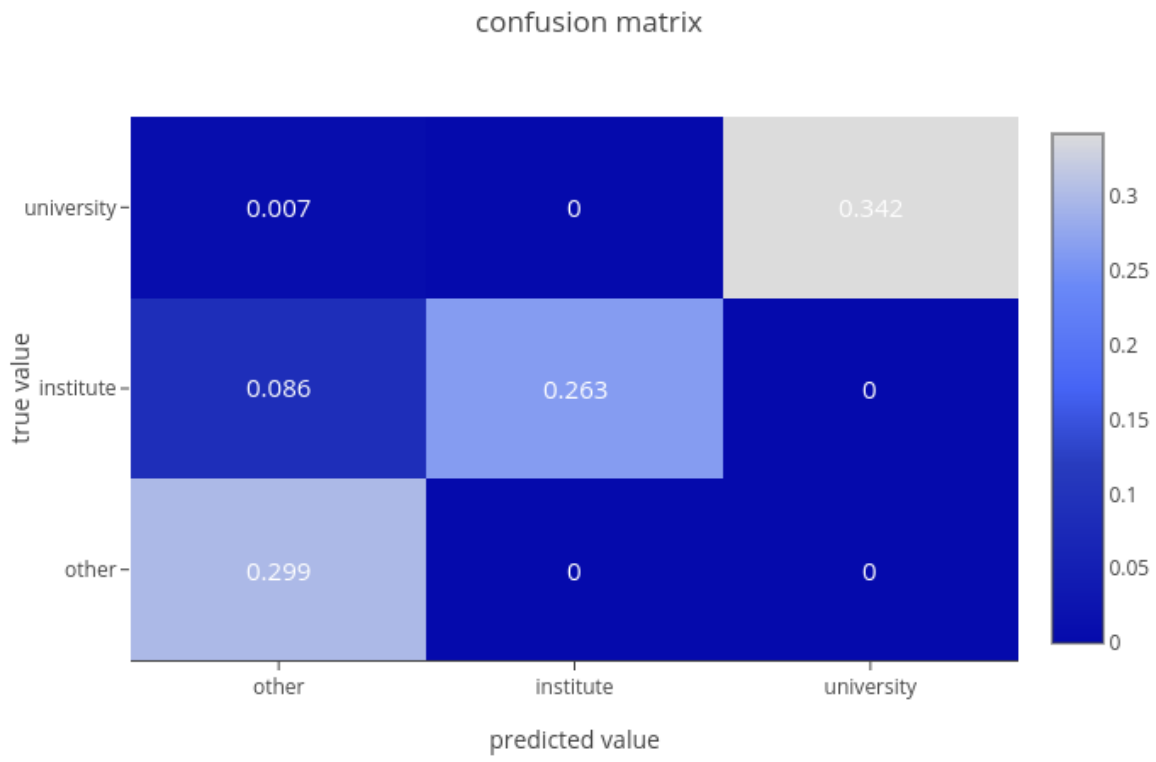

Рисунок 10. Результаты классификации (confusion matrix)

На рисунке 10 представлена так называемая confusion matrix (матрица ошибок), которая показывает сколько объектов класса i были распознаны как объекты класса j. Матрица содержит нормализованные значения, сумма

значений в строке отображает долю сайтов из тестового набора, относящихся к классу, сумма значений в столбце – долю сайтов, отнесённых классификатором к указанному классу. Так, пересечение строки *university* со столбцом *other* показывает, что к классу *other* была отнесена доля 0.007 сайтов класса *university* от общего числа сайтов в тестовом множестве.

В результате исследования общая аккуратность (accuracy) классификации составила 90.6% и, как можно заметить на рисунке 6, основная ошибка – неверная классификация почти трети страниц категории *institute* (более 8% от общего числа) к категории *other*. При этом ни один из веб-сайтов, не относящихся к категориям *institute* и *university*, не был отнесён к ним классификатором. Значения метрик, отражающих качество классификации, приведены в таблице 5. Наилучшие результаты были достигнуты для категории *university*.

Таблица 5. Значения метрик качества классификации для каждой категории

категория	**university**	**institute**	**other**
accuracy	98%	67.3%	100%
recall	100%	100%	76.3%
precision	97.9%	75.3%	100%
F-мера	0.989	0.859	0.865

Заключение

В результате данной работы исследована задача классификации веб-сайтов с учётом зашумлённости и политематичности и реализовано программное решение для разбиения множества веб-сайтов на несколько категорий, удовлетворяющее поставленным условиям. В работе описаны существующие подходы к решению задачи и выбранные методы вместе с их преимуществами и недостатками. Предлагается своё решение данной задачи и подробно описывается алгоритм с его практической реализацией. В конце представлены результаты работы программы на специально подобранном наборе данных. Полученное решение показало свою работоспособность и, в целом, обладает высокой точностью на выбранном датасете. Выявленные недостатки критически оценены, что позволяет определить цели дальнейшего развития данной работы.

В дальнейшем планируется использование большего набора характеристик url адресов и текста для повышения точности классификации, а также модернизировать работу веб-краулера, чтобы уменьшить число ресурсов, ограничивающих ему доступ для загрузки данных. Для удобства взаимодействия пользователя с алгоритмом предполагается представление реализации в виде веб-сервиса с возможностью вывода результатов в различных форматах данных. Также планируется улучшить методы работы алгоритма с памятью вычислительного устройства для повышения производительности на больших объёмах данных.

Список литературы

1. Хомякова Д. Покажи мне свой сайт, и я скажу, кто ты [Электронный ресурс] // Наука в Сибири. http://www.sbras.info/articles/sciencestruct/pokazhi-mne-svoi-sait-i-ya-skazhu-kto-ty
2. Вузы в России [Электронный ресурс] // Федеральный портал «Российское образование». http://www.edu.ru/vuz/
3. Подведомственные организации [Электронный ресурс] // Сайт федерального агенства научных организаций. https://fano.gov.ru/ru/about/sub_organizations/index.php
4. Rukavitsyn A.N., Kupriyanov M.S., Shorov A.V., Petukhov I.V. Investigation of Website Classification Methods Based on Data Mining Techniques // Proceedings of the 19th International conference on soft computing and measurements (SCM 2016). – 2016. – P. 333-336.
5. Маслов М., Пяллинг А., Трифонов С. Автоматическая классификация веб-сайтов // Труды РОМИП, 2007. 6 с.
6. Ji-bin Zh., Zhi-ming X., Kun-li X. и Qi-shu P. A Web Site Classification Approach Based On Its Topological Structure // International Journal on Asian Language Processing, 2010. Vol. 20, No 2. P. 75–86.
7. Page L., Brin S., Motwani R., and Winograd T. The pagerank citation ranking: Bringing order to the web // Stanford InfoLab, 1999.
8. Adsadawut Chanakitkarnchok, Kulit Na Nakorn, Kultida Rojviboolchai. Autonomous website categorization with pre-defined dictionary // 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2016.
9. EasyList. https://easylist.to/
10. Qi X., Davison B. Web page classification: Features and algorithms // ACM Computing Surveys, 2009. Vol. 41, No 2. P 12:1–12:31.

11. Dou Sh., Zheng Ch., Qiang Y., Hua-Jun Z., Benyu Zh., Yuchang L., Wei-Ying M. Web-page Classification through Summarization // ACM SIGIR, 2004.
12. Печников А.А., Д.И. Адаптивный краулер для поиска и сбора внешних гиперссылок // Управление большими системами. Выпуск 36. М.: ИПУ РАН. – 2012. – С.301-315.
13. Korobov M. Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts, 2015. P. 320–332.
14. Левитин А.В. Алгоритмы. Введение в разработку и анализ. М.: Вильямс, 2006. 576 с.
15. EasyList Forum. https://forums.lanik.us/viewforum.php?f=102
16. RIP Google PageRank score: A retrospective on how it ruined the web [Электронный ресурс] // Search Engine Land. https://searchengineland.com/rip-google-pagerank-retrospective-244286
17. Wallach, H.M. Topic modeling: beyond bag-of-words // In Proceedings of the 23rd international conference on Machine learning. ACM, 2006. P. 977-984.
18. Метрики в задачах машинного обучения [Электронный ресурс] // Хабр. https://habr.com/company/ods/blog/328372/
19. Writing Adblock Plus filters [Электронный ресурс] // Adblock Plus. https://adblockplus.org/en/filters
20. HTTP Status Codes [Электронный ресурс] // REST API Tutorial. http://www.restapitutorial.com/httpstatuscodes.html
21. MechanicalSoup. https://mechanicalsoup.readthedocs.io/en/stable/
22. BeautifulSoup. https://www.crummy.com/software/BeautifulSoup/bs4/doc/
23. Pandas. https://pandas.pydata.org/
24. Scikit-learn. http://scikit-learn.org/stable/
25. Plotly. https://plot.ly/
26. BOINC: Fundamental & Applied Science & Technology. http://boincfast.ru/
27. Семинар РОМИП'2009. http://romip.ru/ru/2009/index.html

Приложение 1. Исходный код

Листинг 1. Исходный код main.py

```
import classification
import init
import manager
import scraper
import text
import errno
import numpy
import os
import pandas
import plotly.plotly as py
from colored import fg, attr
from datetime import datetime
from pprint import pprint

title = '🛠 bachelor [' + str(os.getpid()) + ']'
message = 'initializing data...'
# init.URLS for default dataset, init.TEST for tests
dataset = init.URLS
begin = datetime.now()
init.notify(title, message)
try:
    os.makedirs(init.DATA_PREFIX)
except OSError as e:
    if e.errno != errno.EEXIST:
        raise
init.adblock()
py.sign_in(init.PLOTLY['username'], init.PLOTLY['api_key'])

# Get all urls from .txt to array of strings
children_count, initial_data, init.CATEGORIES = init.from_file(dataset)

# Results of scraping: 'url', 'type', 'text'
initial_data['url'] = init.parallel(scraper.get_actual_url,
initial_data['url'], mode='map')
initial_data.to_csv(init.DATA_PREFIX + 'init.csv', sep=',', encoding='utf-8',
index=False)

roots = initial_data.copy()
roots = roots.dropna(subset=['url']).reset_index(drop=True)
roots['root'] = init.parallel(manager.get_root, roots['url'], mode='map')
queue = roots.copy()
roots['children'] = children_count
roots = roots.drop(columns=['url'])
queue['status'] = '-'

message = 'parsing websites...'
init.notify(title, message)
data = pandas.DataFrame()
counter = 0
while not (roots.equals(roots.loc[roots['children'] <= 0]) or
queue.equals(queue.loc[queue['status'] == '+'])) or\
    numpy.array_equal(queue.loc[queue['status'] == '-', 'root'].unique(),
                      roots.loc[roots['children'] <= 0, 'root'].unique()):
```

```
    queue, roots, data = manager.manage(queue, roots, data)
    print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
len(queue))
    pprint(roots)
    roots.to_csv(init.DATA_PREFIX + str(counter) + '_roots.csv', sep=',',
encoding='utf-8')
    queue.to_csv(init.DATA_PREFIX + str(counter) + '_queue.csv', sep=',',
encoding='utf-8')
    counter += 1

message = 'parsing data...'
init.notify(title, message)
data = text.parse_text(data, init.TRAIN_DATA, init.TRAIN_TOKENS)

message = 'classification...'
init.notify(title, message)
classification.classify(data, roots)

# Exit
message = 'script has finished'
end = str((datetime.now() - begin).total_seconds() / 60.0)
end = end.split('.')[0] + '.' + end.split('.')[1][:1]
init.notify(title, message, end + ' minutes spent')
```

Листинг 2. Исходный код init/__init__.py

```
import os
import pandas
import platform
import plotly.plotly as py
import plotly.graph_objs as go
import requests
from colored import fg, attr
from datetime import datetime
from multiprocessing import cpu_count
from multiprocessing.dummy import Pool as ThreadPool


def notify(title, text, subtitle='', sound='Glass'):
    """Send os notification
        :param title:
        :param subtitle:
        :param text:
        :param sound:
    """
    print()
    print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
fg(8) + text + attr(0))
    if subtitle != '':
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
fg(8) + subtitle + attr(0))
    print()

    # macOS notification
    if platform.system() == 'Darwin':
        os.system("""osascript -e 'display notification "{}" with title "{}"
subtitle "{}" sound name "{}"'""".
                  format(text, title, subtitle, sound))
        os.system("""say -v Alex {} {}""".format(text, ' and ' + subtitle if
subtitle else ''))


def parallel(func, parameters, mode='map', threads=cpu_count() - 1):
    """Run function with multithreading"""
    results = None
    # Make the Pool of workers
    with ThreadPool(threads) as pool:
        if mode == 'map':
            results = pool.map(func, parameters)
        if mode == 'starmap':
            results = pool.starmap(func, parameters)

        pool.close()
        pool.join()

    return results


def from_file(file):
    """Get initial data from file
        :param file: path to .txt file
        :return n: num of children pages of each website
        :return data: category - url pairs
```

```
    """
    with open(file, 'r') as input_file:
        n = int(input_file.readline())  # num of sites' subpages needed to be
downloaded (including root)
        data = pandas.read_csv(input_file, sep=" ", header=None,
names=['purpose', 'category', 'url'])
        categories = data.category.unique()

    return n, data, categories


def adblock():
    for rule in ADBLOCK_RULES:
        r = requests.get(rule)

        with open('data/' + rule.rsplit('/', 1)[-1], 'wb') as f:
            f.write(r.content)


def get_output(output, results):
    try:
        results.to_csv(output, sep=',', encoding='utf-8')
    except:
        print(fg(1) + 'something wrong with init.get_output()' + attr(0))


def confusion_matrix(output, confusion_matrix):
    confusion_matrix = 100. * confusion_matrix / confusion_matrix.sum()
    trace = go.Heatmap(z=confusion_matrix,
                       x=CATEGORIES,
                       y=CATEGORIES,
                       colorscale='Blues')
    data = [trace]
    annotations = flatten(
        [
            [{
                "x": j,
                "y": i,
                "font": {
                    "color": "rgb(255, 255, 255)",
                    "size": 15
                },
                "showarrow": False,
                "text": "%.3f" % confusion_matrix[i][j],
                "xref": "x",
                "yref": "y"
            } for i in range(len(CATEGORIES))
            ] for j in range(len(CATEGORIES))
        ]
    )
    layout = go.Layout(
        title="confusion matrix",
        autosize=True,
        dragmode="pan",
        hovermode="closest",
        showlegend=False,
        xaxis={
            "autorange": True,
            "exponentformat": "none",
```

```
            "range": [-0.5, len(CATEGORIES) - 0.5],
            "showgrid": True,
            "showline": True,
            "showspikes": False,
            "showticklabels": True,
            "side": "bottom",
            "tickmode": "auto",
            "ticks": "outside",
            "title": "predicted value",
            "type": "category"
        },
        yaxis={
            "autorange": True,
            "range": [-0.5, len(CATEGORIES) - 0.5],
            "showspikes": False,
            "title": "true value",
            "type": "category"
        },
        annotations=annotations
    )
    fig = go.Figure(data=data, layout=layout)
    py.image.save_as(fig, filename=DATA_PREFIX + output)

    return


PLOTLY = {'username': 'nikitalpopov',
          'api_key': 'RfIZTYe9ndj6humMGTzX'}

INIT_TIME = datetime.now()
INIT_PREFIX = 'init/'
DATA_PREFIX = 'data/{date:%Y-%m-%d_%H:%M:%S}/'.format(date=INIT_TIME)

CATEGORIES = []

URLS = INIT_PREFIX + 'urls.txt'
TEST = INIT_PREFIX + 'test.txt'

TRAIN_DATA = DATA_PREFIX + 'train_data.csv'
TRAIN_TOKENS = DATA_PREFIX + 'train_tokens.csv'
TEST_DATA = DATA_PREFIX + 'test_data.csv'
TEST_TOKENS = DATA_PREFIX + 'test_tokens.csv'

RESULTS = DATA_PREFIX + 'results.csv'
EXCEL = DATA_PREFIX + 'classification.xlsx'

ADBLOCK_RULES = ['https://easylist-
downloads.adblockplus.org/ruadlist+easylist.txt',
'https://filters.adtidy.org/extension/chromium/filters/1.txt']

DEPTH = 1
```

Листинг 3. Исходный код scraper/__init__.py

```
import init
import manager
import ftfy
import lxml.html
import mechanicalsoup
import requests
from adblock import AdRemover
from bs4 import BeautifulSoup, Comment
from colored import fg, attr
from datetime import datetime
from lxml.etree import tostring


def get_actual_url(url):
    try:
        r = requests.get(url, timeout=100)
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
str(r.status_code), url)
    except:
        return None

    if r.url and r.status_code == requests.codes.ok:
        return r.url
    else:
        return None


def parse_url(url):
    if not manager.validate_url(url):

        return {'url': None, 'type': None, 'text': None, 'children': None,
'meta': None, 'title': None}

    browser = mechanicalsoup.StatefulBrowser(
        soup_config={'features': 'lxml'},
        raise_on_404=True
    )
    try:
        response = browser.open(url)
        actual_url = browser.get_url()
    except:
        return {'url': None, 'type': None, 'text': None, 'children': None,
'meta': None, 'title': None}

    print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
          fg(2) + str(response.status_code) + attr(0), actual_url)

    if browser.get_current_page() and response.status_code < 400:
        webpage = browser.get_current_page()
        links = browser.links()

        browser.close()
    else:
        browser.close()

        return {'url': actual_url, 'type': 0, 'text': '', 'children': [],
'meta': '', 'title': ''}
```

```
    page_type = 1
    page_text = ''
    children = []
    meta = ''
    title = ''

    if ("text/html" in response.headers["content-type"]) and webpage:
        # Remove ads
        remover = AdRemover(*['data/' + rule.rsplit('/', 1)[-1] for rule in
init.ADBLOCK_RULES])
        try:
            html = requests.get(url).text
            if html:
                document = lxml.html.document_fromstring(html)
                remover.remove_ads(document)
            else:
                return {'url': actual_url, 'type': 0, 'text': '', 'children':
[], 'meta': '', 'title': ''}
            clean_html = tostring(document).decode("utf-8")
            webpage = BeautifulSoup(clean_html, 'lxml')
        except ValueError:
            print(fg('blue') + '[' + str(datetime.now().time()) + ']' +
attr(0), fg('red') + url + attr(0))
            print('ValueError')
        except TypeError:
            print(fg('blue') + '[' + str(datetime.now().time()) + ']' +
attr(0), fg('red') + url + attr(0))
            print('TypeError')
        except:
            print(fg('blue') + '[' + str(datetime.now().time()) + ']' +
attr(0), fg('red') + url + attr(0))
            print('Some error with removing ads')

        # Remove all comments
        for comments in webpage.findAll(text=lambda text: isinstance(text,
Comment)):
            comments.extract()  # rip it out

        # Remove all script and style elements
        for script in webpage(['script', 'style']):
            script.extract()  # rip it out

        # temp = webpage.find('meta')
        for tag in webpage.find_all("meta"):
            if tag is not None:
                try:
                    meta = ' '.join([meta, tag['content'], tag['keywords'],
tag['description']])
                except KeyError:
                    pass
        temp = webpage.find('title')
        if temp is not None:
            title = temp.text
        body = webpage.find('body')
        page_text = webpage.get_text(" ")

        # Get all children links at webpage
        if body:
```

```
            for link in links:
                children.append(link.get('href'))
            children = [child for child in list(filter(None,
list(set(children)))) if not child.startswith('#')]
            # pprint(children)
        else:
            page_type = 0

        # If needed to fix encoding
        # page_text = ftfy.fix_text(page_text)
        # print(page_text.strip())

        if page_text:
            # Break into lines and remove leading and trailing space on each
            lines = (line.strip() for line in page_text.splitlines())
            # Break multi-headlines into a line each
            chunks = (phrase.strip() for line in lines for phrase in
line.split('  '))
            # Drop blank lines
            page_text = '\n'.join(chunk for chunk in chunks if chunk)
            page_text = page_text.splitlines()
            page_text = list(dict.fromkeys(page_text))
        else:
            page_text = []
            page_type = 0
        page_text = ' '.join(page_text)

    result = {
        'url':      actual_url,
        'children': children,
        'type':     page_type,
        'meta':     meta,
        'title':    title,
        'text':     page_text
    }

    return result
```

Листинг 4. Исходный код manager/__init__.py

```
import init
import scraper
import text
import pandas
import requests
from colored import fg, attr
from datetime import datetime
from urllib import parse


def validate_url(url):
    """Check if url is valid
        :param url: string with URL
        :return bool:
    """
    try:
        url.encode('ascii')
        if url.startswith('mailto:'):
            raise ValueError
        result = scraper.get_actual_url(url)

    except ValueError:
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
              fg(1) + 'ERR' + attr(0), url, fg(1) + 'is invalid url' +
attr(0))
        return False

    except UnicodeEncodeError:
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
              fg(1) + 'ERR' + attr(0), url, fg(1) + 'has bad characters' +
attr(0))
        return False

    except TimeoutError:
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
              fg(1) + 'ERR' + attr(0), url, fg(1) + ':: Operation timed out'
+ attr(0))
        return False

    except ConnectionResetError:
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
              fg(1) + 'ERR' + attr(0), url, fg(1) + ':: Connection reset by
peer' + attr(0))
        return False

    except requests.exceptions.HTTPError as err:
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
              fg(1) + 'ERR' + attr(0), url, fg(1) + str(err) + attr(0))
        return False

    except requests.exceptions.RequestException as err:
        print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
              fg(1) + 'ERR' + attr(0), url, fg(1) + str(err) + attr(0))
        return False

    except:
        return False
```

```
    else:
        if result:
            return True
        else:
            return False


def get_root(url):
    split = parse.urlsplit(url)
    netloc = split.netloc
    path = split.path
    if netloc.startswith('www.'):
        netloc = netloc[4:]
    return (netloc + path.rstrip('/')).split('/')[0]


def get_root_domain(url):
    """Get parents name to fetch with children
        :param url: string with URL
        :return string:
    """
    if url is None:
        return ''

    http = text.find_between(url, 'http://', '/')
    https = text.find_between(url, 'https://', '/')
    if http != '':
        if http.startswith('www.'):
            return http[4:]
        else:
            return http
    else:
        if https != '':
            if https.startswith('www.'):
                return https[4:]
            else:
                return https
        else:
            return ''


def fix_url(url, root):
    """Try to fix children URL
        :param url: string with URL
        :param root: string with root URL
        :return url: fixed accessible url or None
    """
    if root in url:
        if validate_url(url):
            return url
        else:
            if not url.endswith('/'):
                if validate_url(url + '/'):
                    return url + '/'
            if url.startswith('https://'):
                if validate_url(url[:4] + url[5:]):
                    return url[:4] + url[5:]
                else:
```

```
                    return None
            else:
                return None
    else:
        parsed = get_root_domain(url)
        if parsed == '':
            if url.startswith('/'):  # '/link'
                if validate_url(root[:-1] + url):
                    return root[:-1] + url
                else:
                    return None
            else:  # 'link'
                if url.startswith('./'):  # '/link'
                    if validate_url(root + url[2:]):
                        return root[:-1] + url
                    else:
                        return None
                elif validate_url(root + url):
                    return root + url
                else:
                    return None
        else:
            return None


def scrape(url, queue, roots):
    """Scrape data for given url
        :param url:
        :param queue:
        :param roots:
        :return result:
    """
    queue.loc[queue['url'] == url, 'status'] = '+'
    root = get_root(url)
    if len(roots.loc[roots[roots.root == root].index, "children"] > 0) == 1
and \
            (roots.loc[roots[roots.root == root].index, "children"] >
0).values[0]:
        parsed = scraper.parse_url(url)
        if parsed['type'] is not None:
            roots.loc[roots[roots.root == root].index, "children"] -= 1
            return parsed
    return


def fix(child, roots):
    """Try to fix children url (relative or any other) to full-path-url with
info about root and category
        :param child:
        :param roots:
        :return result:
    """
    link, root_url = child
    root = get_root(root_url)
    if len(roots[roots.root == root].children > 0) == 1 and (roots[roots.root
== root].children > 0).values[0]:
        check = fix_url(link, root_url)
        if check != '' and check is not None:
            return (check,
```

```
                    roots.loc[roots[roots.root ==
root].index].root.values[0],
                    roots.loc[roots[roots.root ==
root].index].category.values[0])
    return


def manage(queue, roots, dataframe):
    """Manage data for given websites
        :param queue:
        :param roots:
        :param dataframe:
        :return queue:
        :return dataframe:
    """
    scraped = []
    while len(queue.loc[queue['status'] != '+']) > 0:
        scraped = [x for x in
                   init.parallel(scrape, [(url, queue, roots) for url in
queue.loc[queue['status'] != '+', 'url']],
                                 mode='starmap') if x is not None]

    appendix = pandas.DataFrame.from_records(scraped).dropna(how='all')
    appendix['root'] = ''
    appendix['depth'] = 1. / init.DEPTH
    init.DEPTH += 1

    children = []
    if len(appendix):
        for _, root in roots.iterrows():
            appendix.loc[appendix['url'].str.contains(root.root), 'root'] =
root.root
        dataframe = dataframe.append(appendix, ignore_index=True)

        for root in appendix['url']:
            for links in appendix.loc[appendix.url == root, 'children']:
                for link in links:
                    children.append((link, root))

    fixed = [x for x in init.parallel(fix, [(child, roots) for child in
children], mode='starmap') if x is not None]
    children = pandas.DataFrame\
        .from_records(fixed, columns=['url', 'root', 'category'])\
        .dropna(how='any')\
        .drop_duplicates(subset='url')
    children['status'] = '-'
    queue = queue.append(children, ignore_index=True)
    queue = queue[~queue.duplicated(subset=['url'], keep=False)]
    queue = queue.loc[~queue['root'].isin(roots.loc[roots['children'] <= 0,
'root'].values)]
    dataframe['purpose'] = ''
    dataframe['category'] = ''
    for i in roots.index:
        dataframe.loc[dataframe['root'] == roots['root'][i], 'category'] =
roots['category'][i]
        dataframe.loc[dataframe['root'] == roots['root'][i], 'purpose'] =
roots['purpose'][i]

    return queue, roots, dataframe
```

Листинг 5. Исходный код adblock/__init__.py

```
import cssselect
from colored import fg, attr


class AdRemover(object):
    """
    This class applies elemhide rules from AdBlock Plus to an lxml
    document or element object. One or more AdBlock Plus filter
    subscription files must be provided.

    Example usage:

    >>> import lxml.html
    >>> remover = AdRemover('fanboy-annoyance.txt')
    >>> doc = lxml.html.document_fromstring("<html>...</html>")
    >>> remover.remove_ads(doc)
    """

    def __init__(self, *rules_files):
        if not rules_files:
            raise ValueError("one or more rules_files required")

        translator = cssselect.HTMLTranslator()
        rules = []

        for rules_file in rules_files:
            with open(rules_file, 'r') as f:
                for line in f:
                    # elemhide rules are prefixed by ## in the adblock filter
syntax
                    if line[:2] == '##':
                        try:
                            rules.append(translator.css_to_xpath(line[2:]))
                        except cssselect.SelectorError:
                            # just skip bad selectors
                            pass

        # create one large query by joining them the xpath | (or) operator
        self.xpath_query = '|'.join(rules)

    def remove_ads(self, tree):
        """Remove ads from an lxml document or element object.

        The object passed to this method will be modified in place."""

        for elem in tree.xpath(self.xpath_query):
            elem.getparent().remove(elem)
```

Листинг 6. Исходный код text/__init__.py

```
import init
import pandas
import pymorphy2


def find_between(string, first, last):
    """Find substring between first 'first' and last 'last'
        :param first:
        :param last:
        :return string:
    """
    try:
        if string == '':
            return ''
        start = string.index(first) + len(first)
        end = string.rindex(last, start)

        return string[start:end]

    except AttributeError:
        return ''

    except ValueError:
        return ''


def clear_text(text):
    # Remove all short words
    text = text.str.replace(r'\W*\b\w{1,2}\b', ' ')

    # All text to lower case
    text = text.str.lower()

    # Cleaning text from useless characters
    text = text.str.replace(r'[^\u0400-\u04FF]', u' ')
    text = text.str.replace(' - ', ' ')
    text = text.str.replace('[0-9]', ' ')
    text = text.str.replace('- ', ' ')
    text = text.str.replace(' -', ' ')
    text = text.str.replace(u' . ', ' ')
    text = text.str.replace(u'.', ' ')
    text = text.str.replace(u'[', ' ')
    text = text.str.replace(u']', ' ')
    text = text.str.replace(u'=', ' ')
    text = text.str.replace(u'+', ' ')
    text = text.str.replace(r',!?&*#№@|/():"«»§±`~', ' ')
    text = text.str.replace(u' +', ' ')
    text = text.str.strip()

    return text


def my_tokenizer(s, morph):
    parts_of_speech = ('NUMR', 'PREP', 'CONJ', 'PRCL', 'INTJ', 'NPRO',
'COMP', 'PRED')
    t = s.split(' ')
    f = ''
```

```
    for j in t:
        m = morph.parse(j.replace('.', ''))
        if len(m) != 0:
            wrd = m[0]
            if wrd.tag.POS not in parts_of_speech:
                f = f + ' ' + str(wrd.normal_form)

    return f


def parse_text(dataframe, input, output):
    writer = pandas.ExcelWriter(init.DATA_PREFIX + 'data.xlsx')
    dataframe.to_excel(writer, 'train')
    dataframe.to_csv(input, sep=',', encoding='utf-8', index=False)

    morph = pymorphy2.MorphAnalyzer()
    for column in ['text', 'title', 'meta']:
        dataframe[column] = clear_text(dataframe[column]).apply(lambda x:
my_tokenizer(x, morph))
    dataframe.to_excel(writer, 'tokens')
    dataframe.to_csv(output, sep=',', encoding='utf-8')

    return dataframe
```

Листинг 7. Исходный код classification/__init__.py

```
import init
import pandas
from colored import fg, attr
from datetime import datetime
from pprint import pprint
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.externals import joblib
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.svm import LinearSVC


class ItemSelector(BaseEstimator, TransformerMixin):
    """For data grouped by feature, select subset of data at a provided key.

    The data is expected to be stored in a 2D data structure, where the first
    index is over features and the second is over samples.  i.e.

    >> len(data[key]) == n_samples

    Please note that this is the opposite convention to scikit-learn feature
    matrixes (where the first index corresponds to sample).

    ItemSelector only requires that the collection implement getitem
    (data[key]).  Examples include: a dict of lists, 2D numpy array, Pandas
    DataFrame, numpy record array, etc.

    >> data = {'a': [1, 5, 2, 5, 2, 8],
               'b': [9, 4, 1, 4, 1, 3]}
    >> ds = ItemSelector(key='a')
    >> data['a'] == ds.transform(data)

    ItemSelector is not designed to handle data grouped by sample.  (e.g. a
    list of dicts).  If your data is structured this way, consider a
    transformer along the lines of
`sklearn.feature_extraction.DictVectorizer`.

    Parameters
    ----------
    key : hashable, required
        The key corresponding to the desired value in a mappable.
    """
    def __init__(self, key):
        self.key = key

    def fit(self, x, y=None):
        return self

    def transform(self, data):
        pprint(self.key)
        pprint(data[self.key])
        return data[self.key]


def predict(train, test, model):
    """Train classifier and predict results
```

```
        :param train:
        :param test:
        :param model:
    """
    pipeline = Pipeline([
        ('union', FeatureUnion(
            transformer_list=[
                ('text', Pipeline([
                    ('selector', ItemSelector(key='text')),
                    ('hasher', HashingVectorizer(n_features=2**16)),
                ])),
                ('title', Pipeline([
                    ('selector', ItemSelector(key='title')),
                    ('hasher', HashingVectorizer(n_features=2**16)),
                ])),
                ('meta', Pipeline([
                    ('selector', ItemSelector(key='meta')),
                    ('hasher', HashingVectorizer(n_features=2**16)),
                ]))
            ],
        )),
        ('svc', LinearSVC()),
    ])
    pipeline.fit(train, train.category)
    joblib.dump(pipeline, model)
    tst = pipeline.predict(test)
    result = pandas.Series(tst)

    return result


def assert_class_to_root(dataframes):
    dataframes = dataframes.groupby(['root', 'prediction'])['root']\
        .count().reset_index(name='count').sort_values(['count'],
ascending=False)
    result = dataframes.groupby(['root']).head(1).reset_index(drop=True)

    return result


def classify(dataframe, roots):
    train = dataframe.loc[~dataframe['purpose'].isin(['test'])].copy()
    test = dataframe.loc[dataframe['purpose'].isin(['test'])].copy()

    predicted = predict(train, test, init.DATA_PREFIX + 'model.pkl')

    writer = pandas.ExcelWriter(init.EXCEL)
    categories = pandas.concat((predicted.rename('prediction'),
                                test[['category', 'url',
'root']].reset_index(drop=True)), axis=1)
    categories.to_excel(writer, 'prediction')
    results = assert_class_to_root(categories)
    results = pandas.merge(results, roots[['root', 'category']], on='root')
    print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0),
          'accuracy score: {}'.format(accuracy_score(results.category,
results.prediction)))

    y_true = pandas.Series(results.category)
    y_pred = pandas.Series(results.prediction)
```

```
    print(fg('blue') + '[' + str(datetime.now().time()) + ']' + attr(0))
    pprint(pandas.crosstab(y_true, y_pred, rownames=['True'],
colnames=['Predicted'], margins=True)
           .apply(lambda r: 100.0 * r / r.sum()))
    init.confusion_matrix('confusion_matrix.png',
                          confusion_matrix(results.category,
results.prediction, labels=init.CATEGORIES))
    results.to_excel(writer, 'results')
    init.get_output(init.RESULTS, results)
```